Network Intrusion Detection using Decision Trees

Edward Nataniel C. Apostol
2010-27627 | CS 176 THX
edward.nataniel@yahoo.com

1. INTRODUCTION

Cases of network intrusion attacks are increasing. The difficulty of detecting and countering these attacks are also increasing.

This analysis will try to use a machine learning technique to gain insight on how to detect and counter these attacks.

2. OBJECTIVES

The objective of this analysis is to gain useful insights from the different features that might determine whether a connection is normal or anomalous. Using the obtained information, it will be easier to detect and counter these attacks.

3. METHODOLOGY

A dataset from DARPA was provided. The dataset was obtained from several gigabytes of binary TCP dump data coming from more than a month of network traffic.

The file *kddcup_data_10_percent.csv* contains a subset of millions of connection records from a military environment. The data set has a total of 494021 records and 42 attributes. Among those 41 features are basic features of TCp connections, content features within a connection as suggested by domain knowledge and traffic features computed within a two-second time window. The data in this file will be used to build the classifier.

The file *kddcup_testdata_unlabeled_10_percent.csv* is the test data set. The generated decision tree model will decide on the labels of the unlabelled records.

The file *corrected.csv* is a file downloaded from http://kdd.ics.uci.edu/. It contains the corrected labels of the test data. I downloaded this file from the website to calculate the test accuracy.

The file *pa2.r* is the script written to train the C5.0 decision tree, to generate the rules from the training data and to predict the labels of the unlabelled records.

3.1 Installation of the package

```
1  #installing the c50 package
2  install.packages("C50")
3  library(C50)
```

Figure 1. Installation of the c50 package

For this programming assignment, a package *c50* will be used. This model can be a full decision tree or a collection of rules. It also contains functions that plots the tree, predicts new samples, and summarizes the c5.0 model.

3.2 Reading and storing the training data

```
12  #reading the csv file. storing the data the dataframe mydata.
13  #adding column names to the dataframe
14  trainData = read.csv("kddcup_data_10_percent.csv",header=FALSE)
15  colnames(trainData) <- c("duration","protocol_type","service","
17  #x is the dataframe of predictors.
18  #y is the factor vector
19  x <- trainData[,1:41]
20  y <- trainData[,42]
```

Figure 2. Reading and storing the training data

The training data will come from the file named *kddcup_data_10_percent.csv*. The variable *x* will store the predictors or the first 41 columns of the data while the variable *y* is the vector of labels.

3.3 Training the decision tree

```
22  #trains the C5.0 decision tree to generate the node "treeModel"
23  treeModel <- C50::C5.0(x,y)
24  summary(treeModel)
```

Figure 3.Training the decision tree

The decision tree will be trained by calling the *c5.0* function from the c50 package. It will take some time for the training to be complete because of the huge amount of training data. The *summary* function prints out the detailed summary of the c5.0 model. It prints the structure of the tree, evaluation on the training data, and the attribute usage in percent. This function can help us determine the most predictive attributes.

3.4 Computing for the classification accuracy for the training set

The classification accuracy for the training set can be computed by predicting the labels of the training set and comparing the predicted labels to the actual labels from the original dataset.

```
26  #getting training_accuracy
27  predictedLabels <- predict(treeModel, x, type = "class")
28  trainingAccuracy <- sum(as.character(predictedLabels)==as.character(y))/length(y)
29  trainingAccuracy #prints the trainingAccuracy
```

Figure 4.Computing the accuracy for the training set

Predicting labels can be done by using the *predict* function from the c50 package. Line 28 of the code counts the number of matches between the predicted labels and the actual labels.

3.5 Generation of rules

```
36    treeModel2 <- C50::C5.0(x,y,rules=TRUE)
37    summary(treeModel2)
```

Figure 5. Generating the set of rules from the training data

The set of rules can be generated from the Training Data by just calling the c50 function and adding the argument *rules=TRUE*. By calling the *summary* function again, the list of rules will be printed.

3.6 Predicting the labels from test data

The *predict* function from the c50 package was used. The test data includes type of attacks not from the training data.

```
46    #reading testData and adding column labels
47    testData = read.csv("kddcup_testdata_unlabeled_10_percent.csv",he
48    colnames(testData) <- c("duration","protocol_type","service","fla
49
50    #predictedLabels2 are the predicted values of the unlabeled test
51    predictedLabels2 <- predict(treeModel, testData, type = "class")
```

Figure 6.Predicting labels from the test data

The confidence values of the prediction can also be displayed by changing the *type* argument from *class* to *prob*. Almost all of the confidence values of the prediction are above **99.9%**

3.7 Computing for the classifaction accuracy for the test data

Since the actual labels of the test data was not given, I downloaded the *corrected.csv* from *http://kdd.ics.uci.edu* to calculate for the classification accuracy of the model to the test data set.

```
53    #true labels of the test data. This is not originally incluc
54    #i downloaded the file from http://kdd.ics.uci.edu/ to get t
55    testDataCorrected <- read.csv("corrected.csv",header=FALSE)
56    colnames(testDataCorrected) <- c("duration","protocol_type",
57    testDataCorrected <- testDataCorrected[,42]
59  #comparing the labels of the predicted values and the actual values.
60  #accuracy = number of matching labels / number of instances
61  testAccuracy <- sum(as.character(predictedLabels2)==as.character(testDataCorrected))/length(test
62  testAccuracy #prints the accuracy
```

Figure 7.Computing for the test accuracy

Predicting labels can be done by using the *predict* function from the *c50* package. Line 61 of the code counts the number of matches between the predicted labels and the actual labels.

4. EXPERIMENTAL RESULTS

The classification accuracy for the given training set is **99.9684%** while the training accuracy for the given testing dataset is **91.6358%.** Also, the confidence values of the prediction are above **99.9%.**

The *summary* functions prints out a detailed summary of the c5.0 model It includes the attribute usage or the importance of the attribute. The importance is represented by the percentage of records that fall under that fall into terminal nodes after the split.

```
Attribute usage:

100.00% wrong_fragment
 99.75% service
 80.13% src_bytes
 42.84% same_srv_rate
 41.76% dst_host_diff_srv_rate
 20.93% dst_host_serror_rate
 20.70% num_compromised
 20.48% flag
 20.08% dst_host_srv_diff_host_rate
 20.00% num_failed_logins
 19.79% hot
 19.77% root_shell
 19.71% srv_serror_rate
  1.50% dst_host_same_src_port_rate
  1.23% dst_host_count
  0.86% dst_host_srv_count
  0.85% duration
  0.73% rerror_rate
  0.30% serror_rate
  0.30% protocol_type
  0.22% land
  0.15% count
  0.14% logged_in
  0.14% dst_bytes
  0.14% num_root
  0.04% dst_host_same_srv_rate
  0.04% diff_srv_rate
  0.03% srv_count
  0.02% num_shells
  0.02% num_file_creations
```

Figure 8.Attribute usage for treeModel

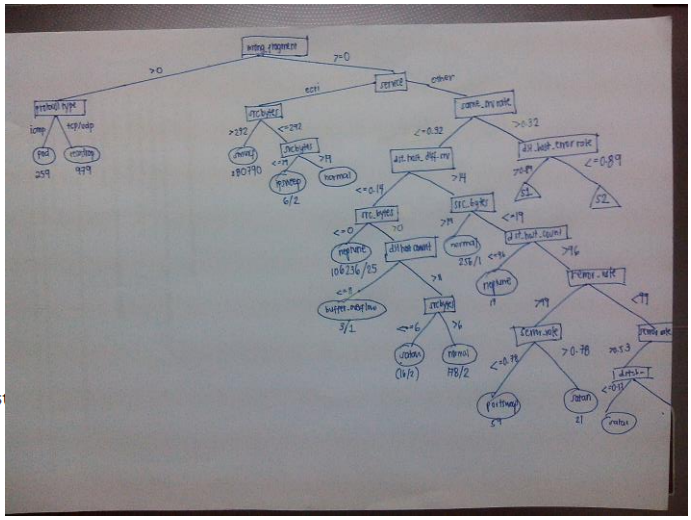

Figure 9. The upper part of the trained decision tree

5. ANALYSIS AND DISCUSSION OF RESULTS

The accuracy of the classifier is quite high for both the training and testing set.

.

The number of wrong fragments *wrong_fragment*, network service on the destination *service*, and the number of data bytes from the source to the destination *src_bytes* are the the most important attributes or the major predictors. They greatly affect the classification of records.

The number of file creations operation num_file_creations, land (whether the connection comes from the same host/port) and number of shell prompts *num_shell* are the least important attributes.

Some interesting rules with very high confidence are:

1) If the number of source bytes is greater than 10073 and there exist a compromised condition, then the connection is automatically a back attack (a type of denial of service attack)
2) If the service is in any of the ff {co_i, ftp, gopher, link, mtp, name, private, remote_job, rje, ssh, time} and the dst_host_srv_diff_host_rate > 0.48, the connection is an ipsweep (a type of probing attack)
3) If the flag/error status of the connection is SH and the number of connections to the same service as the current connection in the past two seconds is less than or equal to 80, then the connection is an nmap (a type of probe attack)

6. CONCLUSION

Machine learning techniques like decision trees can be used to detect these intrusions. They can learn what patterns are normal and what patterns are unusual with a large accuracy. Decision trees generate a model that classifies whether a connection is normal connection or an attack. Rules with high confidence are also very useful.